STAT

Vulnerable to TOCTOU issues

Sean Barnum, Cigital, Inc. [vita[1]]

2007-04-17

Part "Original Cigital Coding Rule in XML"

Mime-type: text/xml, size: 8040 bytes

Attack Category	• Path spoofing or confusion problem
Vulnerability Category	• Indeterminate File/Path • TOCTOU - Time of Check, Time of Use
Software Context	• File Management
Location	
Description	The stat() function obtains information about the file pointed to by path. Read, write or execute permission of the named file is not required, but all directories listed in the path name leading to the file must be searchable. Lstat() is like stat() except in the case where the named file is a symbolic link, in which case lstat() returns information about the link, while stat() returns information about the file the link references. fstat() obtains the same information about an open file known by the file descriptor fd. stat() (in combination with other functions that manipulate the file being queried; e.g., mkdir) is vulnerable to TOCTOU attacks. A call to stat() should be flagged if the first argument (the directory name) is used later in a use-category call.
APIs	**Function Name** / **Comments** _stat / check; win32 _tstat / check; win32 _wstat / check; win32 lstat / check stat / check
Method of Attack	The key issue with respect to TOCTOU vulnerabilities is that programs make assumptions about atomicity of actions. It is assumed that checking the state or identity of a targeted resource

1. http://buildsecurityin.us-cert.gov/bsi/about_us/authors/35-BSI.html (Barnum, Sean)

	followed by an action on that resource is all one action. In reality, there is a period of time between the check and the use that allows either an attacker to intentionally or another interleaved process or thread to unintentionally change the state of the targeted resource and yield unexpected and undesired results. The stat() call is a check-category call, which when followed by a use-category call can be indicative of a TOCTOU vulnerability.
Exception Criteria	
Solutions	

Solution Applicability	Solution Description	Solution Efficacy
Generally applicable to all uses of stat.	Consider using a safer approach to using stat such as that outlined below for opening and creating files as outlined in Building Secure Software (referenced below), page 220. 1) Perform an lstat() of the file before opening it, saving the stat structure. 2) Perform Open(), passing the O_CREAT an O_EXCL flags which will cause the open to fail if the file cannot be created. 3) Perform an fstat() on the file descriptor returned by the open() call, saving the stat structure. 4) Compare three fields in teh two stat structures to be sure they	Effective

		are equivalent: st_mode, st_info & st_dev. If these comparisons are successful, then we know the lstat() call happened on the file we ultimately opened. Moreover, we know that we did not follow a symbolic link (which is why we used lstat() instead of stat()). (This solution comes from the book Building Secure Software referenced below) Always verify that the stat function successfully accessed the file and loaded the description in the struct.	
	Generally applicable.	The most basic advice for TOCTOU vulnerabilities is to not perform a check before the use. This does not resolve the underlying issue of the execution of a function on a resource whose state and identity cannot be assured, but	Does not resolve the underlying vulnerability but limits the false sense of security given by the check.

		it does help to limit the false sense of security given by the check.	
	Generally applicable.	Limit the interleaving of operations on files from multiple processes.	Does not eliminate the underlying vulnerability but can help make it more difficult to exploit.
	Generally applicable.	Limit the spread of time (cycles) between the check and use of a resource.	Does not eliminate the underlying vulnerability but can help make it more difficult to exploit.

Signature Details	int stat(const char *path, struct stat *sb);
	int lstat(const char *path, struct stat *sb);
	int fstat(int fd, struct stat *sb);

Examples of Incorrect Code	`char filename="theFile.txt";`
	`struct stat statBuffer;`
	`stat(fileName,&statBuffer );`
	`In this case, the information concerning the file will be placed in statBuffer. If there are other functions being called, it is possible that they may attempt to access the file at the same time. Also, if there is a command immediately after that assumes successful filling of the buffer, there can be a problem.`

	`#include <sys/types.h>`
	`#include <sys/stat.h>`
	`int check_status;`
	`int use_status;`
	`struct stat statbuf;`
	`...`
	`check_status=stat("tobecreateddir", &statbuf);`
	`...`
	`<long enough intervening code>`

	`use_status=mkdir("tobecreateddir",..);`

Examples of Corrected Code	One solution is to eliminate the pre-creation test and instead use a post-creation status check.
	`#include <sys/types.h>`
	`#include <sys/stat.h>`
	`int status;`
	`...`
	`status = mkdir("/home/cnd/mod1", S_IRWXU | S_IRWXG | S_IROTH | S_IXOTH);`
	`...`

Source References	• Viega, John & McGraw, Gary. *Building Secure Software: How to Avoid Security Problems the Right Way*. Boston, MA: Addison-Wesley Professional, 2001, ISBN: 020172152X, pg. 220. • UNIX man page for stat() • Microsoft Developer Network Library (MSDN)
Recommended Resource	
Discriminant Set	**Operating Systems**: • Windows • UNIX **Languages**: • C • C++

Cigital, Inc. Copyright

1. mailto:copyright@cigital.com